

Assignment #1 Procedures

- Create a program that will:
 - Create a `typedef` for the exit status information returned from the `wait()` call

```
typedef union{
    int exit_status;
    struct{
        unsigned sig_num:7;
        unsigned core_dmp:1;
        unsigned exit_num:8;
    }parts;
}LE_Wait_Status;
```

- Create a **signal handler function** to be inherited by a child process (this function will load the Assign1.c program)
- Create a **pipe** using the `pipe()` system call to be inherited by a child process
- Print out its own credentials as shown in the on-line source code Assign1.c

Assignment #1 Procedures (cont'd)

- Your program will then:
 - **Create** a child process using the `fork()` call
 - The parent will then read the **read-channel** of the pipe with the `read()` system call, waiting for the child to write to it with the `write()` system call
 - When the parent **awakes** from the pipe read it will **send signal SIGTERM (#15)** using the `kill()` system call to the child process (the child will now be in its endless loop)
 - The parent now **awaits** the death of the child in the `wait()` system call
 - When the child dies, the parent prints out the **child's exit status** and then exits itself

Assignment #1 Procedures (cont'd)

- The **child** process that you create will:
 - Come into existence on the **return** side of the **fork()** call that the parent used to create it
 - Set up the **signal handler function** using the **sigaction()** system call to catch the signal the parent will send (SIGTERM)
 - Collect and print out its credentials as did the parent
 - Write a character into the pipe using the **write()** call to wake up the parent after its credentials are printed
 - Enter an **endless loop**, expecting the arrival of the parent signal to force it to execute the signal handler function (there should be a timeout in this loop to terminate in the event the signal doesn't arrive)

Assignment #1 Procedures (cont'd)

- The **child**, upon catching the parent's signal, will now:
 - Use the `exec1()` call to load in the executable built from the source code **Assign1.c** (this executable can be found on mercury at: `~bill/cs308/A1` or you can just build one yourself from the `Assign1.c` source code on-line)
 - When the `Assign1.c` program is **loaded into the child process**, it will collect and print its credentials out and also print a message asking the user to enter the **kill command** from the command line, providing the **correct PID** number for you to use with the `kill` command to force **termination** of this process now running the `Assign1.c` program

Assignment #1 Procedures (cont'd)

- The Assign1.c program will then:
 - Enter an **endless loop**, expecting the SIGTERM signal to arrive when the user types the **kill** command
 - When this program gets the SIGTERM signal it will run its handler which will call `exit()` to terminate
 - When the **child terminates**, the parent will awake from its `wait()` call and report how the child terminated (**exit or signal, core or no-core**)
 - The parent program will then do a **normal exit** after reporting on the child's termination status

Expected Output

This is the Parent process report:

PARENT PROG: Process ID is: 135329

PARENT PROG: Process parent ID is: 87149

PARENT PROG: Real UID is: 1004

PARENT PROG: Real GID is: 4000

PARENT PROG: Effective UID is: 1004

PARENT PROG: Effective GID is: 4000

PARENT PROG: Process priority is: 0

PARENT PROG: will now create child, read pipe,
signal child, and wait for obituary
from child

PARENT PROG: created Child with 135259 PID

Expected Output (cont'd)

This is the Child process report:

CHILD PROG: Process ID is: 135259

CHILD PROG: Process parent ID is: 135329

CHILD PROG: Real UID is: 1004

CHILD PROG: Real GID is: 4000

CHILD PROG: Effective UID is: 1004

CHILD PROG: Effective GID is: 4000

CHILD PROG: Process priority is: 0

CHILD PROG: about to write pipe and go to endless loop

PARENT PROG: read pipe and sent SIGTERM, will now wait
for child to exit

CHILD PROG: Awake in handler - You Got Me With Signal
Number 15 after spinning for more than
1373021 loop iterations

CHILD PROG: Now beginning to exec Prof program, goodbye

Expected Output (cont'd)

This is the Prof process report:

PROF PROG: Process ID is: 135259

PROF PROG: Process parent ID is: 135329

PROF PROG: Real UID is: 1004

PROF PROG: Real GID is: 4000

PROF PROG: Effective UID is: 1004

PROF PROG: Effective GID is: 4000

PROF PROG: Process priority is: 5

PROF PROG: going into and endless loop,
use kill 135259 to kill me now

<SHELL COMMAND> kill 135259

PROF PROG: Awake in handler - You Got Me With Signal
Number Number 15 after spinning for more
than 12 Billion loop iterations, goodbye

PARENT PROG: Child 135259 exited with exit code 1, bye

Assignment #1 Procedures (cont'd)

- System calls you will need:

- `getpid()`
- `getppid()`
- `getuid()`
- `geteuid()`
- `getgid()`
- `getegid()`
- `getpriority()`
- `read()`
- `write()`
- `sigprocmask()`
- `sigaction()`
- `fork()`
- `execl()`
- `kill()`
- `wait()`
- `pipe()`